Auto-scaling Techniques for Elastic Data Stream Processing

Thomas Heinze, Valerio Pappalardo, Zbigniew Jerzak, Christof Fetzer

March 2014

Outline

1. Introduction
2. Auto-scaling Techniques for Elastic Data Stream Processing
3. Evaluation
4. Conclusion and Future Work

Utilization within Cloud Environments

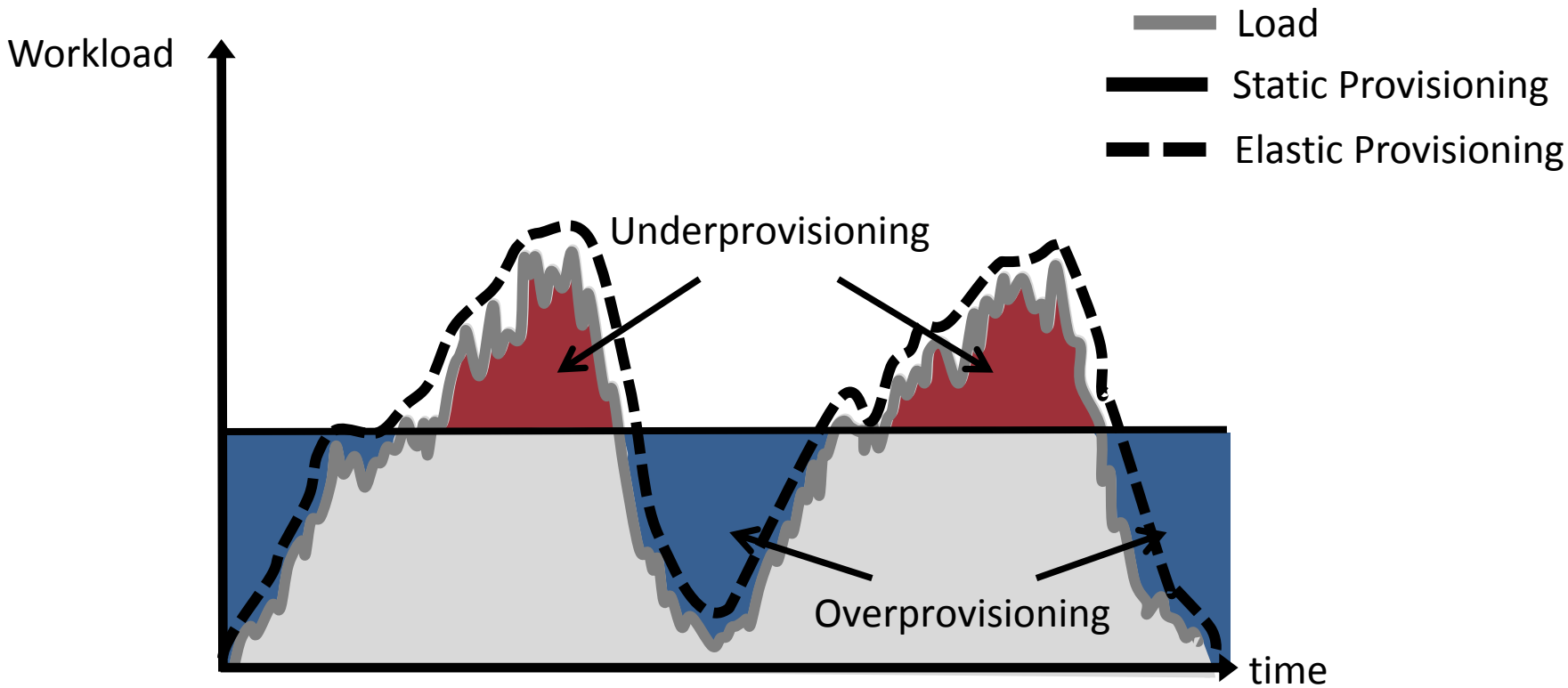
- Cluster of Twitter[1] has average CPU utilization < 20%, however ~80% of the resources are reserved
- Google Cluster Trace[2] shows an average 25-35% CPU utilization and 40% memory

The average utilization within public clouds is estimated to be between 6% to 12%[3].

[1] Benjamin Hindman, et al. "Mesos: A platform for fine-grained resource sharing in the data center." NSDI, 2011.
[2] Charles Reiss, et al. "Heterogeneity and dynamicity of clouds at scale: Google trace analysis." SOCC, ACM, 2012.
[3] Arunchandar Vasan, et al. “Worth their watts? An empirical study of datacenter servers“. HPCA, 2010.

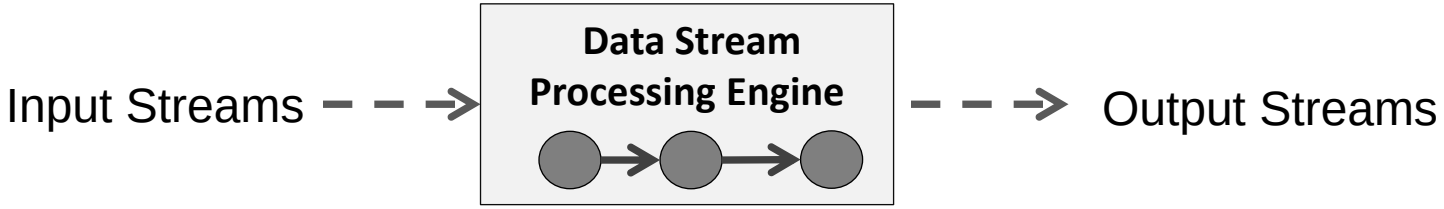
Elasticity

- Users needs to reserve required resources
  - Limited understanding of the performance of the system
  - Limited knowledge of characteristics of the workload



Elastic Data Stream Processing

- Long standing continous queries over potential infinite data stream



✓ Small memory footprint (MB – GB) for most use cases, fast scale out

✗ Strict requirements on end to end latency

✗ Unpredictable workload, high variability (within seconds to minutes)

✗ Load balancing influences running queries

Auto-scaling Techniques[1]

Different algorithmic approaches for various domains and use cases.

1) Threshold-based approaches
2) Time series analysis
3) Reinforcement learning
4) Queuing theory
5) Control theory

[1]T. Lorido-Botran et al., “Auto-scaling techniques for elastic applications in cloud environments,” Tech. Rep., 2012.

Requirements

- Workload Independence: Independent from workload characteristics; we make no assumption on the input workload.
- Adaptivity: Adapt online to changing conditions like different workload characteristics.
- Configurability: Easy to setup and configure by an end user.
- Computational feasibility: The algorithm has to be computationally feasible for scale out within seconds.

Feasible: *Threshold-based approaches*, *Reinforcement Learning*, Control Theory.

Not feasible: Time Series Analysis, Queuing Models.

Threshold-based Approach

- Upper Threshold:

  „If CPU utilization of host is larger than *x* for *y* seconds, host is marked as overloaded.“

- Lower Threshold:

  „If CPU utilization of host is small than z for *w* seconds, host is marked as underloaded.“

- Additional parameters: Target Utilization, Grace Period

- Two Variants: Local Thresholds vs. Global Thresholds

Reinforcement Learning[1]

- Lookup table describing for each state the best action
- Best action is adapted based on an online learning algorithm

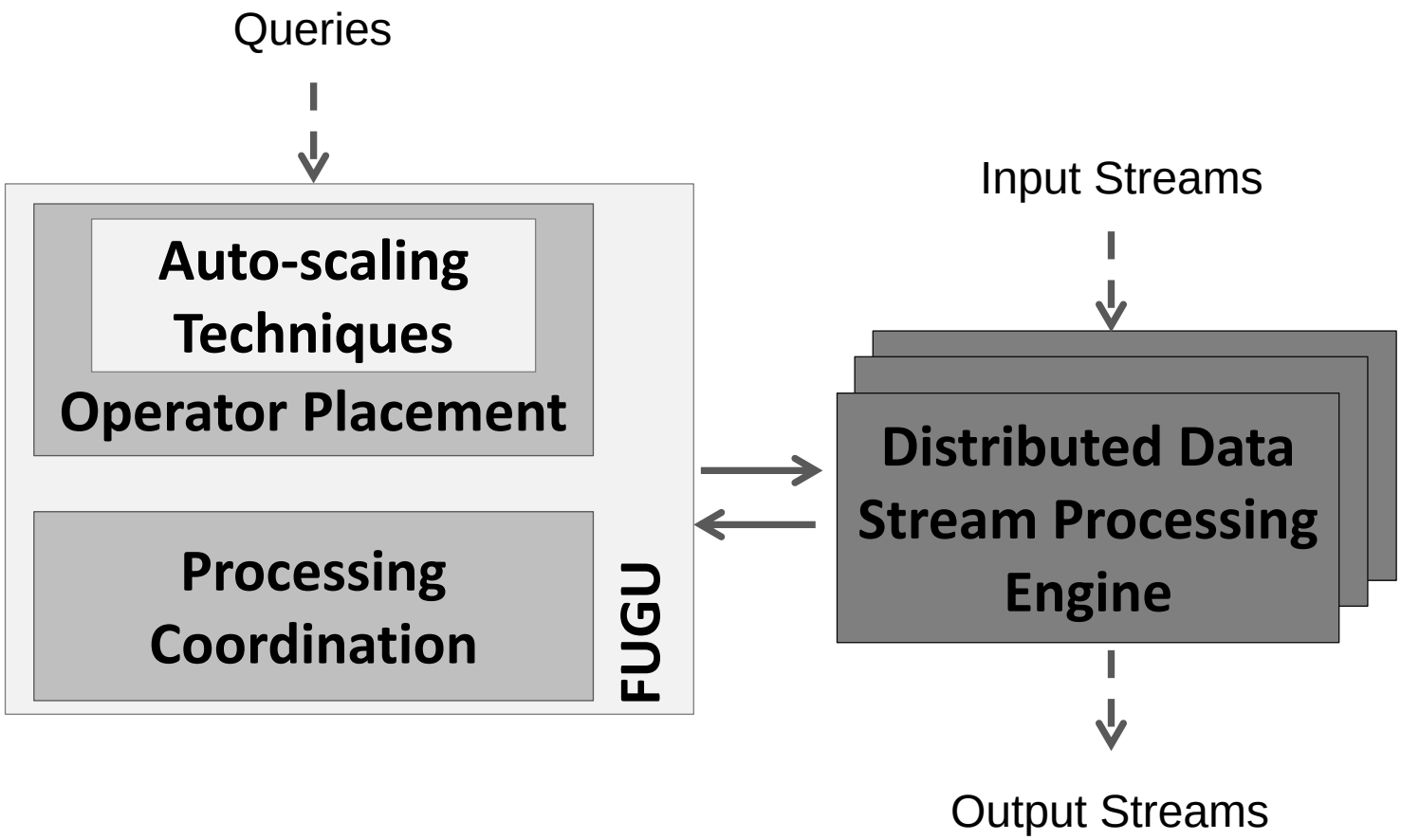
Util	ScaleIn	NA	ScaleOut
0.7	0.4	0.85	0.8
0.8	0.38	0.7	0.85
0.9	0.35	0.5	0.9
1.0	0.32	0.4	1.0

Extension for Elastic Data Stream Processing:

- Local policy
- Initialisation based on threshold-based policy
- Grace Period

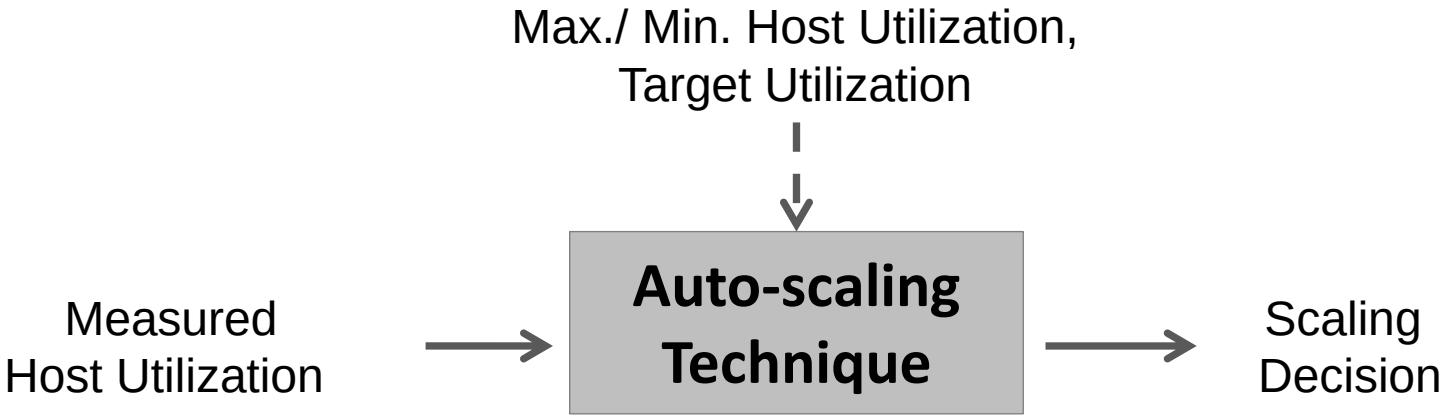
[1]R. Das et al., "*Model-based and model-free approaches to autonomic resource allocation*", IBM Research Report, 2005.

Elastic Data Stream Processing



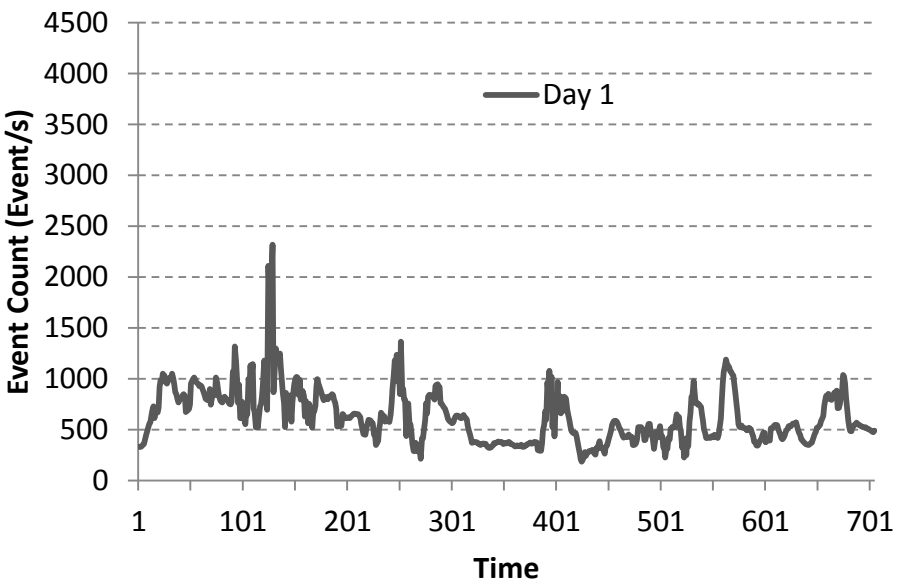
Heinze, Thomas, et al. "*Elastic Complex Event Processing under Varying Query Load.*" *BD3,VLDB, 2013*.

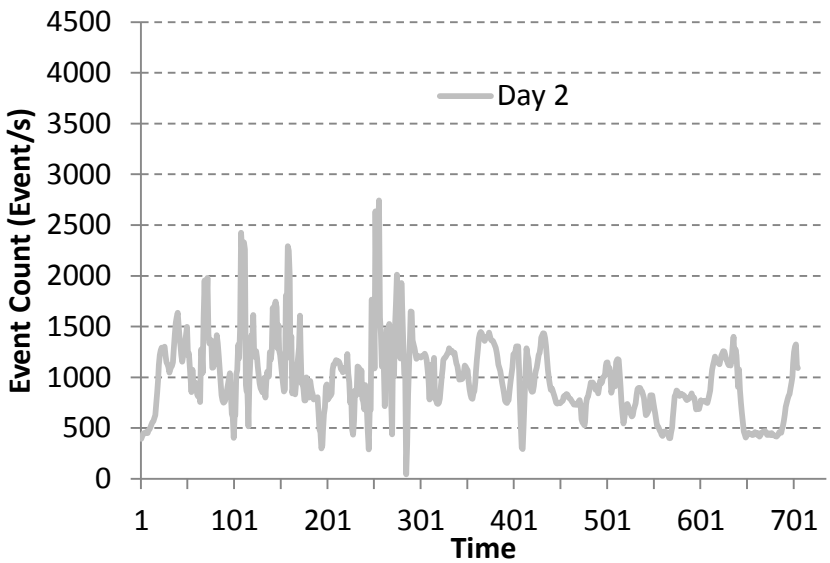
Integration of Auto-Scaling Techniques



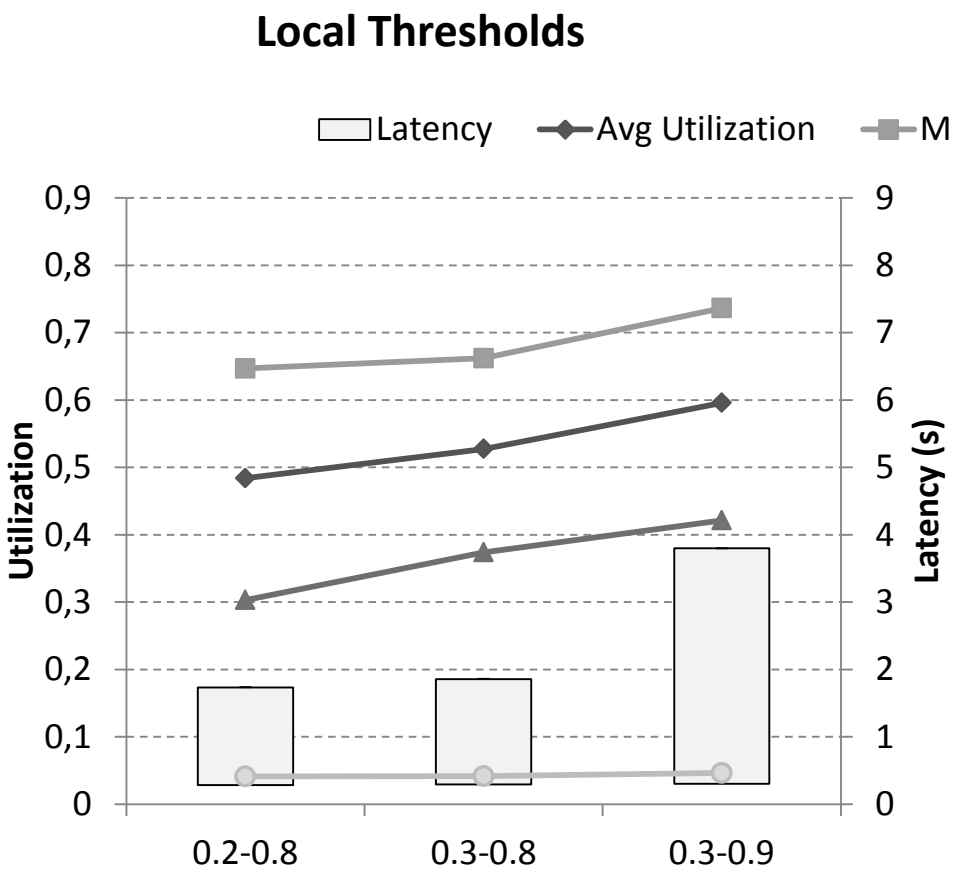
Setup

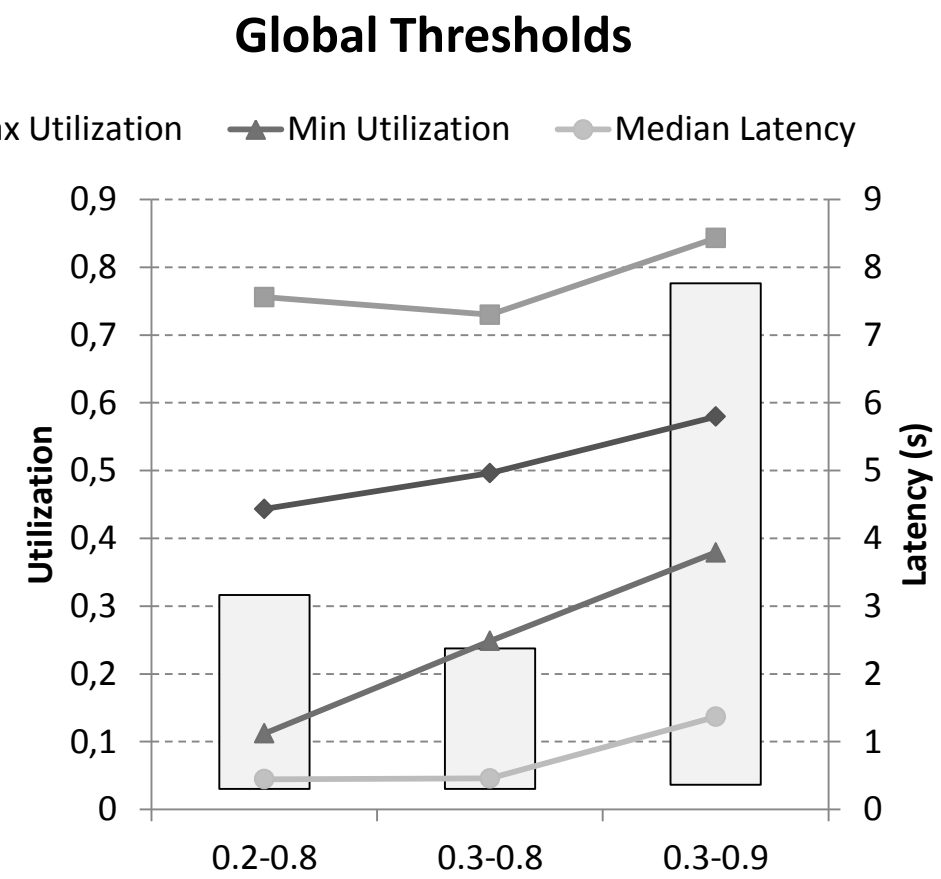
- Based on data taken from Frankfurt Stock Exchange
- 35 aggregation queries using varying data selection and time ranges
- up to 10 processing nodes
- Experiment duration: ~1 hour





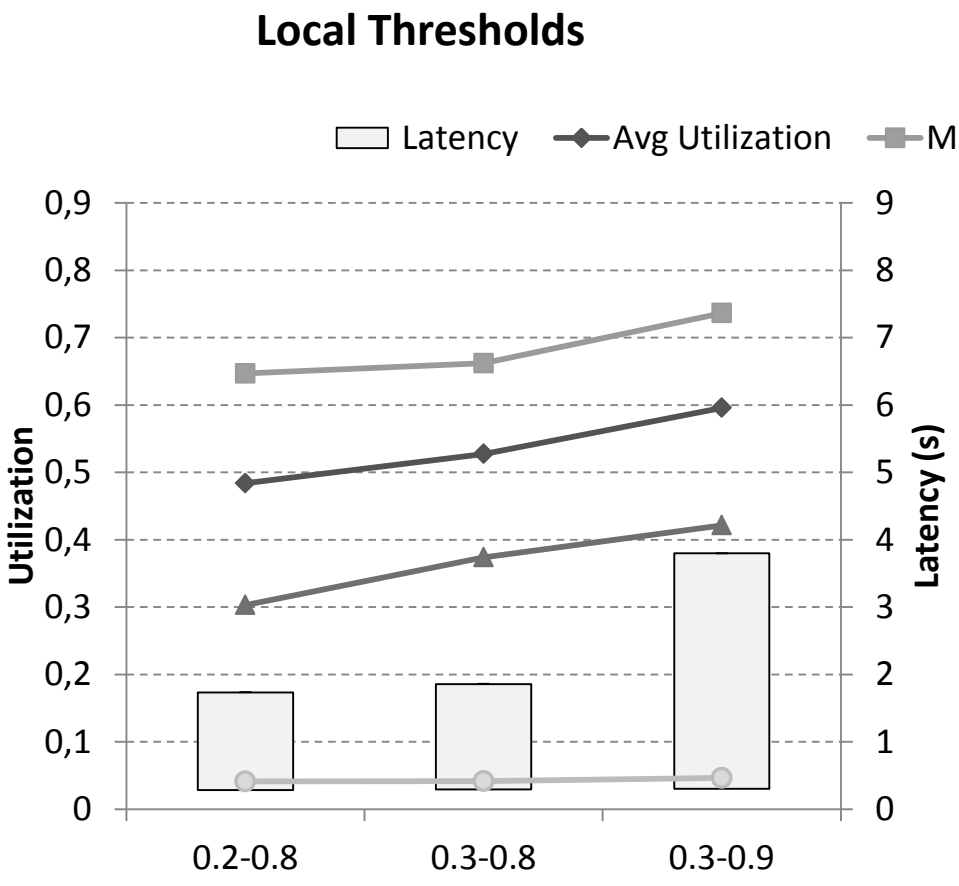
Configurability

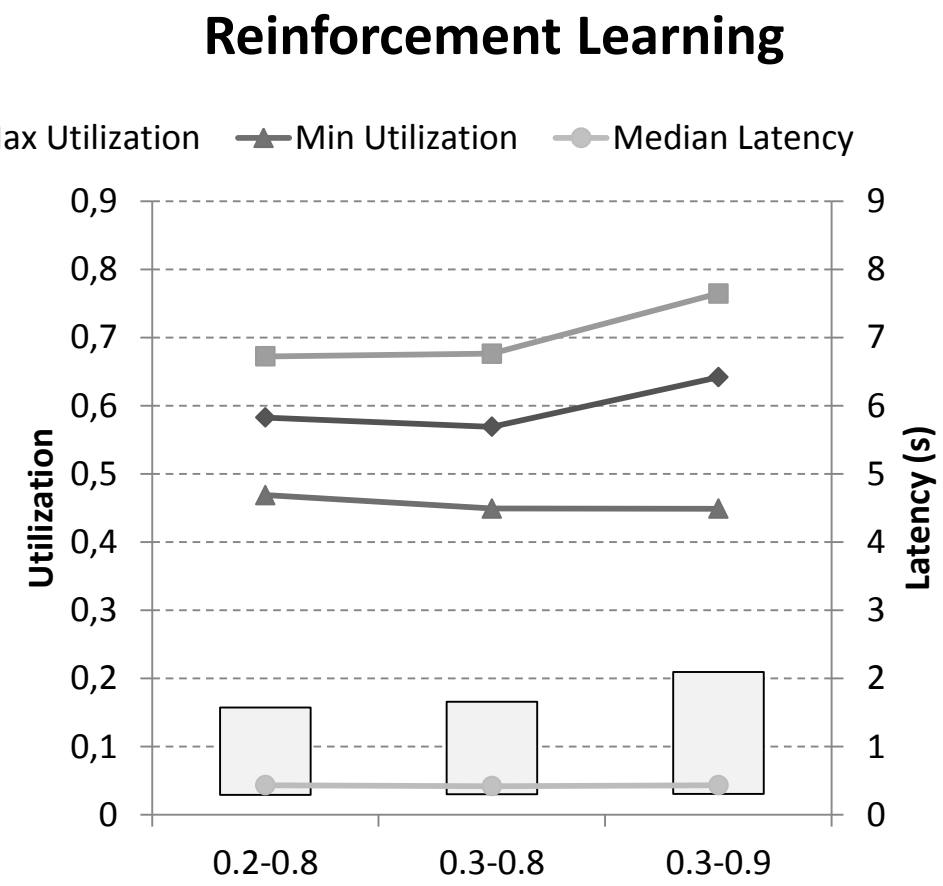




Local threshold are more robust, while global thresholds highly sensitiv.

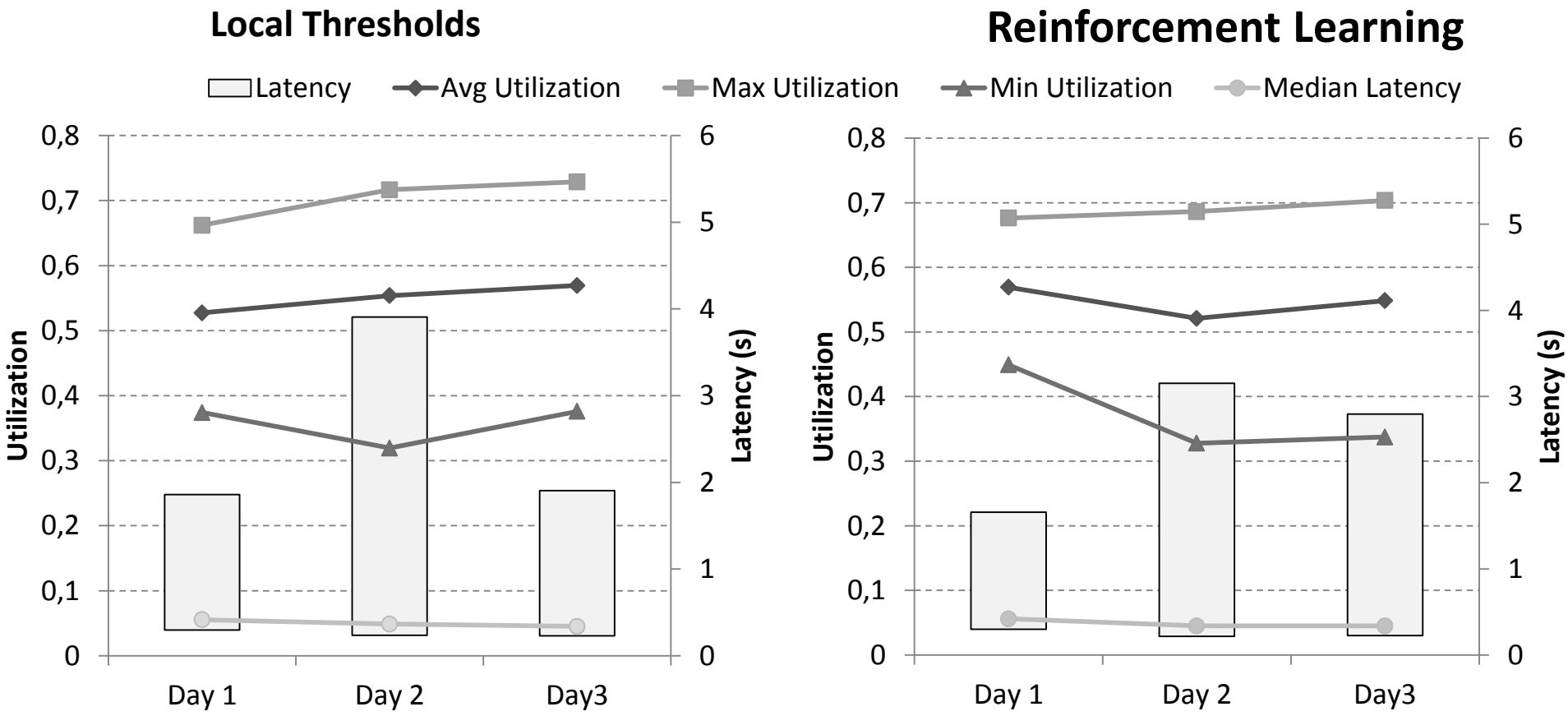
Configurability





Reinforcement Learning achieves best utilization and latency values.

Different Workloads



Reinforcement Learning reduces variability between different workloads.

Lessions Learned

- Elasticity for data stream processing systems poses new challenges towards auto-scaling techniques
- Global thresholds create many overload situations
- Adaptive Reinforcement Learning is more stable than Local Thresholds, improves utilization by 5% and reduces latency by 30%

- Need to handle variations within a workload better
- Investigate additional parameters like latency, queue length.
- Study influence of the placement algorithm